Your search matched **17** of **750881** documents.
Results are shown **25** to a page, sorted by **publication year** in **descending** order.
You may refine your search by editing the current search expression or entering a new one the tex
Then click **Search Again.**

sandbox*

Search Again

Results:
Journal or Magazine = **JNL** Conference = **CNF** Standard = **STD**

1 **Active routing**
Maxemchuk, N.F.; Low, S.H.
Selected Areas in Communications, IEEE Journal on , Volume: 19 Issue: 3 , M 2001
Page(s): 552 -565

[Abstract] [PDF Full-Text (152 KB)] **JNL**

2 **Intranet security with micro-firewalls and mobile agents for proact intrusion response**
Gangadharan, M.; Kai Hwang
Computer Networks and Mobile Computing, 2001. Proceedings. 2001 Interna Conference on , 2001
Page(s): 325 -332

[Abstract] [PDF Full-Text (775 KB)] **CNF**

3 **Micro-firewalls for dynamic network security with distributed intru detection**
Kai Hwang; Gangadharan, M.
Network Computing and Applications, 2001. NCA 2001. IEEE International S on , 2001
Page(s): 68 -79

[Abstract] [PDF Full-Text (1071 KB)] **CNF**

4 **The Mercury Project: a feasibility study for Internet robots**
Goldberg, K.; Gentner, S.; Sutter, C.; Wiegley, J.

IEEE Robotics & Automation Magazine , Volume: 7 Issue: 1 , March 2000
Page(s): 35 -40

[Abstract] [PDF Full-Text (312 KB)] **JNL**

---

5 **Specifying runtime environments and functionalities of downloadab components under the sandbox model**
Kaiya, H.; Kaijiri, K.
Principles of Software Evolution, 2000. Proceedings. International Symposium 2000
Page(s): 138 -142

[Abstract] [PDF Full-Text (340 KB)] **CNF**

---

6 **A flexible access control service for Java mobile code**
Corradi, A.; Montanari, R.; Lupu, E.; Sloman, M.; Stefanelli, C.
Computer Security Applications, 2000. ACSAC '00. 16th Annual Conference ,
Page(s): 356 -365

[Abstract] [PDF Full-Text (832 KB)] **CNF**

---

7 **A language-based approach to programmable networks**
Wakeman, I.; Jeffrey, A.; Owen, T.
Open Architectures and Network Programming, 2000. Proceedings. OPENARC 2000 IEEE Third Conference on , 2000
Page(s): 128 -137

[Abstract] [PDF Full-Text (712 KB)] **CNF**

---

8 **A tool for securely integrating legacy systems into a distributed environment**
Souder, T.; Mancoridis, S.
Reverse Engineering, 1999. Proceedings. Sixth Working Conference on , 199
Page(s): 47 -55

[Abstract] [PDF Full-Text (100 KB)] **CNF**

---

9 **New Millennium DS2 electronic packaging an advanced electronic p "sandbox"**
Arakaki, G.; D'Agostino, S.
Aerospace Conference, 1999. Proceedings. 1999 IEEE , Volume: 2 , 1999
Page(s): 205 -213 vol.2

[Abstract] [PDF Full-Text (1256 KB)] **CNF**

10 **Mobile code security**
Rubin, A.D.; Geer, D.E., Jr.
IEEE Internet Computing , Volume: 2 Issue: 6 , Nov.-Dec. 1998
Page(s): 30 -34

[Abstract] [PDF Full-Text (112 KB)] **JNL**

11 **Security control for COTS components**
Qun Zhong; Edwards, N.
Computer , Volume: 31 Issue: 6 , June 1998
Page(s): 67 -73

[Abstract] [PDF Full-Text (148 KB)] **JNL**

12 **Simulation of thermal contrast on solar radiated sand surfaces con buried minelike objects**
Sjokvist, S.; Georgson, M.; Ringberg, S.; Loyd, D.
Detection of Abandoned Land Mines, 1998. Second International Conference (Conf. Publ. No. 458) , 1998
Page(s): 115 -119

[Abstract] [PDF Full-Text (420 KB)] **CNF**

13 **On certifying mobile code for secure applications**
Ghosh, A.K.
Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on , 1998
Page(s): 381

[Abstract] [PDF Full-Text (64 KB)] **CNF**

14 **Security in the large: is Java's sandbox scalable?**
Qun Zhong; Edwards, N.
Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Sympos 1998
Page(s): 387 -392

[Abstract] [PDF Full-Text (36 KB)] **CNF**

15 **Fuzzy-cluster representation of time-frequency signatures as a me automatic classification of buried mine-like targets**
Strifors, H.C.; Gustafsson, A.; Abrahamson, S.; Gaunaurd, G.C.
Time-Frequency and Time-Scale Analysis, 1998. Proceedings of the IEEE-SP

International Symposium on , 1998
Page(s): 597 -600

[Abstract] [PDF Full-Text (428 KB)] **CNF**

16 **Desktop teleoperation via the World Wide Web**
Goldberg, K.; Mascha, M.; Gentner, S.; Rothenberg, N.; Sutter, C.; Wiegley,
Robotics and Automation, 1995. Proceedings., 1995 IEEE International Confe
Volume: 1 , 1995
Page(s): 654 -659 vol.1

[Abstract] [PDF Full-Text (1032 KB)] **CNF**

17 **The SANDBOX: a virtual reality interface to scientific databases**
Johnson, A.; Fotouhi, F.
Scientific and Statistical Database Management, 1994. Proceedings., Sevent
International Working Conference on , 1994
Page(s): 12 -21

[Abstract] [PDF Full-Text (836 KB)] **CNF**

L7 ANSWER 1 OF 10 USPATFULL

TI System and method for securing a program's execution in a network environment

PI US 6351816 B1 20020226

RLI Division of Ser. No. US 1996-652703, filed on **19 May 1996**

SUMM Written in a general purpose language such as Java.RTM., an applet is in this way unrestrained in its functionality. It can perform any function which a program written in any other general purpose language (such as C or PL1) can accomplish. The methodologies of applets, however, are constrained by the Java.RTM. environment in order to minimize the security risks an applet presents to the workstation 150. That is to say, an applet is restricted to "play" within a bounded "**sandbox** ."

DETD Because servers are typically accessed orders of magnitude more frequently than a client workstation, maintaining the integrity of the server executing a servlet becomes even more critical than maintaining the integrity of the client executing an applet. Corruption on a server can spread quite rapidly to any number of clients. Should corruption pass among servers, the rate of corruption of clients can increase exponentially. The **sandbox** of the servlet is appropriately restricted.

DETD In the HotJava.RTM. browser mentioned above, the boundaries of the **sandbox** for an applet are as follows: An applet may not read, write or inquire into the status of any filesystem on the client workstation 150. An applet from an server, say, 120b running on the workstation 150 can not access any other processor 120a, 120n, 150 over the network 160 other than its server processor 120b. An applet cannot load a library from either its server processor 120 or the workstation 150. An applet cannot initiate the execution of a process. An applet cannot examine the properties of any resource on the workstation 150.

DETD To assist in the enforcement of the boundaries of the **sandbox** of an applet, the assignee of the instant invention has developed a suite of protocols for the development and execution of applets: the Java.RTM. development environment (or Java.RTM. Development Kit). The development environment includes a number of packages ("lang," "io," "net," "util," "awt" and "applet"). To the extent an applet needs language, I/O, network, utility, windowing or application support, the application must resort to the methods available through the classes provided by one or some of the packages of the Java.RTM. development environment.

NCL NCLM: **713/201.000**

L7 ANSWER 2 OF 10 USPATFULL
TI Security enhancement for untrusted executable code
PI US 6275938 B1 20010814
AI US 1997-919844 19970828 (8) <--
AB Untrusted executable code programs (applets or controls) are written in native, directly executable code. The executable code is loaded into a pre-allocated memory range (**sandbox**) from which references to outside memory are severely restricted by checks (sniff code) added to the executable code. Conventional application-program interface (API) calls in the untrusted code are replaced with translation-code modules (thunks) that allow the executable code to access the host operating system, while preventing breaches of the host system's security. Static links in the code are replaced by calls to thunk modules. When an API call is made during execution, control transfers to the thunk, which determines whether the API call is one which should be allowed to execute on the operating system.

SUMM The present invention implements a security policy for untrusted executable code written in native, directly executable code. The executable code is loaded into a pre-allocated memory range, or **sandbox**, from which references to outside memory are restricted. Checks ("sniff code") added to the executable code enforces these restrictions during execution. Conventional application-program interface (API) calls in the untrusted code are replaced with translation-code modules ("thunks") that allow the executable code to access the host operating system, while preventing breaches of the host system's security. Static links in the control or applet are replaced by calls to thunk modules. When an API call is made during execution, control transfers to the thunk, which determines whether the API call is one which should be allowed to execute on the operating system or not.

DRWD FIG. 4 is a simplified block diagram of a **sandbox** area in memory.

DETD When an applet such as 362 is to be executed, a host program 36 such as an Internet web browser invokes emulator 39. The emulator employs its own loader module 396 to load the applet code into a predetermined memory area, and to assign another predetermined memory area for its use. These areas are called the "**sandbox**" for that applet. During execution of the applet, emulator 39 compiles the applet's code in a compiled cache which resides outside the **sandbox**. During the compilation p

7 ANSWER 3 OF 10 USPATFULL

TI System and method for securing a program's execution in a network environment

PI US 6263442 B1 20010717

AI US 1996-652703 19960530 (8) <--

SUMM Written in a general purpose language such as Java, an applet is in this way unrestrained in its functionality. It can perform any function which a program written in any other general purpose language (such as C or PL1) can accomplish. The methodologies of applets, however, are constrained by the Java environment in order to minimize the security risks an applet presents to the workstation 150. That is to say, an applet is restricted to "play" within a bounded "**sandbox**."

DETD Because servers are typically accessed orders of magnitude more frequently than a client workstation, maintaining the integrity of the server executing a servlet becomes even more critical than maintaining the integrity of the client executing an applet. Corruption on a server can spread quite rapidly to any number of clients. Should corruption pass among servers, the rate of corruption of clients can increase exponentially. The **sandbox** of the servlet is appropriately restricted.

DETD In the HotJava browser mentioned above, the boundaries of the **sandbox** for an applet are as follows: An applet may not read, write or inquire into the status of any filesystem on the client workstation 150. An applet from an server, say, 120b running on the workstation 150 can not access any other processor 120a, 120n, 150 over the network 160 other than its server processor 120b. An applet cannot load a library from either its server processor 120 or the workstation 150. An applet cannot initiate the execution of a process. An applet cannot examine the properties of any resource on the workstation 150.

DETD To assist in th

L7 ANSWER 4 OF 10 USPATFULL

TI Method and apparatus for providing security for servers executing application programs received via a network

PI US 6167522 20001226

AI US 1997-829990 19970401 (8) <--

SUMM A method and apparatus for providing security for a server executing programs received by the server via a network is disclosed. An application program that is to be provided by a Web server along with a source identifier is received by the Web server via a network, such as the Internet. Before loading the application program, the server performs a verification procedure including granting access privileges based on the source identifier. Access privileges are granted or withheld for a plurality of resources available to the server. If an application program is received from a known hostile source or no access privileges are granted, the applications program may be rejected. Thus, the resources defining the application program's universe, or **sandbox**, is determined individually based on source identifiers.

DETD Before loading the application program, the server performs a verification procedure including granting access privileges based on the source identifier. Access privileges are granted or withheld for resources available to the server. If an application program is received from a known hostile source, or if no access privileges are granted, the applications program may be rejected. Thus, the resources defining the application program's universe, or **sandbox**, are determined individually based on source identifiers.

DETD Once the byte-codes have been verified, they are translated into servlets configured to run on the architecture of Web server 150. The servlet is then loaded into the servlet's **sandbox** (i.e., a known, bounded area in memory and granted access only to those specific resources listed in the ACL). Once the servlet is loaded in Web server 150, Web browser 170 may access the servlet and any resources available to the servlet. Thus, resource privileges are granted on a servlet-by-servlet basis, which increases the flexibility of a Web server's security. This improved flexibility allows administrators to grant more privileges to known and trusted sources, while granting fewer privileges to new or unknown sources. By eliminating an all-or-nothing security approach, the web may offer more resources in a more convenient manner.

DETD If no access privileges are granted in step 320, the servlet is rejected or not loaded in step 330. If the servlet is granted access privileges in step 320, a **sandbox** is defined for the servlet in step 340. A **sandbox** defines the set of access privileges granted to the servlet.

DETD In step 350, the servlet is loaded by Web server 150. In addition to limitations imposed on a servlet by the **sandbox** to which it is assigned, other checks may be performed. For example, accesses to certain resources may be monitored by the server to guard against hostile behavior.

DETD Thus, by providing identity-based access controls, a Web server can control which servlets have access to which data. This arrangement provides protection against theft or alteration of data. In addition, by restricting a servlet to

L7 ANSWER 6 OF 10 USPATFULL
TI Applet redirection for controlled access to non-orginating hosts
PI US 5987523 19991116
AI US 1997-868611 19970604 (8) <--
SUMM A significant restriction of such applets is that the standard Java model only allows the applets to talk to the servers they were downloaded from. This is referred to as the Java "**sandbox**" security restriction. It provides some security benefits but also severely restricts Java use for some applications. For example, this is undesirable for applets whose main purpose is connectivity where the goal is to accomplish communications with many other systems in the network (networking applets). Recent Java releases such as the Java Development Kit (JDK) version 1.1 provide a solution to this called trusted applets, but this solution does not work for all scenarios. First, it does not address users of prior JDK versions such as 1.02. Second, leading web browsers have yet to fully comply with JDK 1.1. Third, and most importantly, network administrators do not want their users to connect to any arbitrary host in their network. Instead they want the flexibility of multi-host applet communication with the advantage of administrative control and security capabilities. No solution is available which provides all of these advantages.

DRWD FIG. 1 depicts a basic network environment with **sandbox** restriction (Prior Art).
DETD The example network shown in FIG. 1 represents networking applet capabilities with the present Java **sandbox** security restriction. Using a Web client (101) such as a Java-enabled web browser or Java applet viewer, a user runs applets (102) dynamically downloaded from a Web server (103). However, due to the **sandbox** restriction the applet can only communicate with that originating server (103). Applications and resources on the server (104) can be used by the applet to store information or help complete its tasks, but the applet is not allowed to access other servers that may be on the network. As previously mentioned, recent advancements in the Java standard define the framework for trusted applets that can access other resources, but this approach is not widely supported by today's browsers and, more importantly, it does not provide administrative control over what resources an applet can access.
DETD FIG. 3 shows users A and B on Web clients (301 & 306) that use the present invention's redirector (303) function to access a host server (305). For clari

L7 ANSWER 7 OF 10 USPATFULL
TI Security monitor
PI US 5974549 19991026
AI US 1997-825102 19970327 (8) <--
AB The present invention is a method of creating a secure **sandbox** within which a plurality of downloaded software components can execute in a secure manner. The software components can be of any type, e.g., Java, ActiveX, Netscape plugin, etc. The invention implements a security monitor that is injected to the address space of an arbitrary monitored application such as a Web browser, e.g., Internet Explorer, Netscape Navigator, etc. The monitored application then executes in a secure mode in which every software component downloaded executes in a secure **sandbox**. The security monitor detects when such a software component is downloaded and is operative to create the **sandbox** around it before it is permitted to execute. If the software component attempts to commit an action that breaches security, it halts the software component's execution and issues a warning to the user. The security monitor detects attempted security breaches by the software component in accordance with a user configurable security policy. Such a policy may include limiting file read/write access, access to directories, disk access, creation and the reading/writing of network connections, access to system resources and services and access to the address spaces of other processes.
SUMM These security implications are known and different approaches have been taken to solve them. The Java programming language and environment were designed from the ground up with security in mind. Java applets execute in what is termed a secure `**sandbox**,` which is a run time environment in which applets are prevented from executing certain actions. For example, Java applets are not permitted to access local storage, modify system parameters or to establish a network connection to an untrusted site.

SUMM The present invention is a method of creating a secure **sandbox** within which a plurality of downloaded software components can execute in a secure manner. The software components can be of any type, e.g., Java, ActiveX, Netscape plugin, etc. The invention implements a security monitor that is injected to the address space of an arbitrary monitored application such as a Web browser, e.g., Internet Explorer, Netscape Navigator, etc. The monitored application then executes in a secure mode in which every software component downloaded executes in a secure **sandbox**. The security monitor detects when such a download of a software component occurs and is operative to create the **sandbox** around it before it is permitted to execute. If the software component attempts to commit an action that breaches security, it halts the software component's execution and issues a warning to the user.

L7 ANSWER 9 OF 10 USPATFULL
TI Code certification for network transmission
PI US 5892904 19990406
AI US 1996-761484 19961206 (8) <--
SUMM One approach to addressing this problem is to create a protective and padded virtual machine on the software recipient's computer. Such a virtual machine, which is often referred to as a playpen or **sandbox**, allows untrusted, possibly malicious code to be executed without fear that it could cause any unauthorized or unwarranted actions. This approach is an outgrowth of the security architecture in existing computer operating systems. A problem with this approach is that it is extraordinarily difficult to create a **sandbox** that is actually secure against malicious code. Unexpected security holes are commonly discovered in supposedly secure operating systems that use this method.

SUMM But even assuming that this difficulty could be overcome, a fundamental quandary with the sandboxing approach is that there is a very strong tension between creating a **sandbox** safe enough to run perhaps malicious code, but yet with sufficient access to system resources to be capable of performing useful operations. For example, **sandboxed** code that is allowed to make network connections off of a host machine (e.g., TCP, FTP, EMail, or otherwise) should not have access to any information on the machine that is to be kept private. As other examples, some system utilities such as a disk defragmenter or an indexing utility that locates the lost documents on a hard disk would likely be inoperable as **sandboxed** code. A **sandbox** that successfully protected against the damage these utilities might possibly cause would prevent them from carrying out their intended purpose.

NCL NCLM: **713/201.000**
NCLS: **713/170.000; 713/187.000**

```
=> d his

     (FILE 'HOME' ENTERED AT 15:49:23 ON 01 MAR 2002)

     FILE 'USPATFULL' ENTERED AT 15:49:31 ON 01 MAR 2002
L1             0 S ESAFE# OR SECURE4U
L2           202 S SANDBOX##
L3         28823 S 380/?/NCL OR 709/?/NCL OR 717/?/NCL OR 713/?/NCL
L4            37 S L2 AND L3
L5       2559438 S AD<=19970925
L6        600258 S RLD<=19970925
L7            10 S L4 AND (L5 OR L6)
L8          4703 S LEARN?(8A)(APPLICATION# OR CODE# OR INSTRUCTION# OR SOFTWARE#
L9             5 S L2 AND L8
L10            0 S L9 AND (L5 OR L6)
L11          466 S L8 AND L3
L12          357 S L11 AND (L5 OR L6)

=>
```

```
=> d his

     (FILE 'HOME' ENTERED AT 15:49:23 ON 01 MAR 2002)

     FILE 'USPATFULL' ENTERED AT 15:49:31 ON 01 MAR 2002
L1             0 S ESAFE# OR SECURE4U
L2           202 S SANDBOX##
L3         28823 S 380/?/NCL OR 709/?/NCL OR 717/?/NCL OR 713/?/NCL
L4            37 S L2 AND L3
L5       2559438 S AD<=19970925
L6        600258 S RLD<=19970925
L7            10 S L4 AND (L5 OR L6)
L8          4703 S LEARN?(8A)(APPLICATION# OR CODE# OR INSTRUCTION# OR SOFTWARE#
L9             5 S L2 AND L8
L10            0 S L9 AND (L5 OR L6)
L11          466 S L8 AND L3
L12          357 S L11 AND (L5 OR L6)
L13         2104 S MONITOR?(8A)(APPLICATION# OR PROGRAM# OR SOFTWARE# OR APPLET#
L14          514 S L3 AND L13
L15          366 S L14 AND (L5 OR L6)
L16          129 S SECUR?(P)L13
L17           47 S L16 AND L3
L18           36 S L17 AND (L5 OR L6)

=>
```

L18 ANSWER 11 OF 36 USPATFULL

TI Method and system for mounting a system partition as a logical drive while an operating system is operational by modifying a partition table

PI US 5974517 19991026

AI US 1996-710360 19960917 (8) <--

DETD The executive services system 602 includes the set of services including I/O manager 610, object manager 612, **security** reference monitor 614, process manager 616, local procedure call facility 618, and virtual memory manager 620. These services 610-620 are the interface between user-mode environment subsystems and the kernel 608. The I/O manager 610 manages all input and output for the Windows NT operating system 600. The object manager 612 provides uniform rules for retention, naming and **security** of objects. The **security** reference **monitor** 614 ensures that **applications** cannot **access** system resources without authorization. The process manager 616 manages the creation and deletion of processes. The local procedure call facility 618 manages local procedure calls (LPC) which involves message passing between applications and the environment subsystems. The virtual memory manager 620 manages the translation of virtual addresses to physical pages in memory.

NCL NCLM: 711/173.000
NCLS: 711/112.000; **713/001.000**; **713/100.000**

L18 ANSWER 14 OF 36 USPATFULL
TI Method for security shield implementation in computer system's software
PI US 5925126 19990720
AI US 1997-820290 19970318 (8) <--
SUMM The present invention uses software methodology to place a **security** shield in front of any computer system **software** by placing controls and **monitors** in front of all **access** paths to the computer system software. The present invention utilizes computer software to implement a method software call interception technique to provide a simple yet **secure** method to place controls and monitors in access paths of computer system resources. The present methodology protects and **monitors** **access** to resources such as files, directories, **programs** , operator commands, systems and network services, all without modifying the operating system or system binaries. The two call interception techniques provide controls for both operating system requests, such as UNIX system calls, and interactive commands, such as telnet, rsh, and ftp. The present method provides a **security** methodology for open systems that is transparent to the users and is done without direct modification to the underlying operating system and environment, while providing adequate **security**, which is difficult to remove, and which is comprehensive and simple to implement and manage.

DETD An advance **security** monitor (ASM) module for audits and monitors of selective operation system events may be configured to audit an monitor selected activities based on users, processes, programs, and targeted files on designated systems. For example, ASM can be used to **monitor** all privileged **access** from Superusers and "setuid root" **programs** and processes, or to **monitor** all **accesses** to a sensitive file or **program**.
NCL NCLM: **713/200.000**
NCLS: **713/201.000**